Available online at www.sciencedirect.com

SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 324–335

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

A performance model of non-deterministic particle transport on large-scale systems

Mark M. Mathis[a,b], Darren J. Kerbyson[a,*], Adolfy Hoisie[a]

[a] *Performance and Architecture Lab (PAL), CCS-3, Los Alamos National Laboratory, PO Box 1663, Los Alamos, NM 87545, USA*
[b] *Department of Computer Science, Texas A&M University, 301 Harvey R. Bright Building, College Station, TX 77843-3112, USA*

Available online 28 December 2004

Abstract

In this work we present a predictive analytical model that encompasses the performance and scaling characteristics of a non-deterministic particle transport application, MCNP (Monte Carlo N-Particle), that represents part of the Advanced Simulation and Computing (ASC) workload. MCNP can be used for the simulation of neutron, photon, electron, or coupled transport, and has found uses in many problem areas including nuclear reactors, radiation shielding, and medical physics. Monte Carlo methods in general and MCNP specifically do not solve an explicit equation, but rather obtain answers by simulating the interactions between individual particles and a predefined geometry. This is in contrast to deterministic transport methods, the most common of which is the discrete ordinates method, that solve the transport equation directly for the average particle behavior.

Previous studies on the scalability of parallel Monte Carlo calculations have been rather general in nature. The performance model developed here is both detailed and parametric with both application characteristics (e.g. problem size), and system characteristics (e.g. communication latency, bandwidth, achieved processing rate) serving as input. The model is validated against measurements on an AlphaServer ES40 system showing high accuracy across many processor/problem combinations. The model is then used to provide insight into the achievable performance that should be possible on systems containing thousands of processors and to quantify the impact that possible improvements in sub-system performance may have. In addition, the impact on performance of modifying the communication structure of the code is also quantified.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Performance modeling; Performance analysis; Large-scale systems

1. Introduction

MCNP is a general purpose Monte Carlo N-Particle code that represents part of the Advanced Simulation and Computing (ASC) workload. It can be used for the simulation of neutron, photon, electron, or coupled transport [3]. Particle transport simulation has found uses in many problem areas including nuclear reactors, radiation shielding, and medical physics. There is currently great interest in the use of non-deterministic particle simulation on large-scale systems—both those

* Corresponding author.
E-mail address: djk@lanl.gov (D.J. Kerbyson).

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.11.018

that currently exist as well as future advanced systems being proposed. In this work we develop a detailed analytical performance model of MCNP. A performance model of an application can be used for much more than just predicting the time that will be required to run the application on a particular system. It can also be used "in reverse" to determine what size problem can be solved in a fixed time allocation on a particular system. Performance models can also assist users in selecting appropriate systems to use, or acquire, that are suited to their applications [9], or to detect problems in the installed system if the actual times do not match the predictions [10]. A performance model can also be used to identify bottlenecks in the code and to make recommendations for its future development.

We primarily take an analytical approach to performance modeling based on task graphs which seek to capture the performance characteristics of an application through an examination of its key data structures and computational flows. The model consists of two fundamental parts: a computation model and a system model. The computation model represents the computation activities of the actual application, or more precisely the part of the application that consumes the bulk of the execution time. The computation model is based on a static analysis of the key portions of the code but is parameterized in terms of the data specific to the problem being simulated. The system model encapsulates key system characteristics such as communication (e.g. network latency and bandwidth) and computational performance (e.g. processor speed). The two parts of the model are kept separate so the model can be re-used without alteration to explore a multitude of performance scenarios. For instance, one may evaluate a different problem by setting the appropriate input parameters to the computation model, or evaluate a new machine by changing the system model. A similar modeling approach has been used to model other large-scale applications including deterministic transport on structured and unstructured meshes [6,8,13,14], and adaptive mesh refinement [7,9]. A third part of a performance model can be identified corresponding to the data on which the application operates. In this work, we have made the simplifying assumption that all data is local to the processors at the time of computation, thus alleviating the need for a separate data model. In general, however, this will not be the case and the data must be considered independently.

Criticality may be defined as that state of a nuclear chain-reacting medium when the nuclear fission chain reaction just becomes self-sustaining (critical). Criticality safety is a vital part of the storage, transportation, and processing of fissionable materials. MCNP includes the capability to calculate eigenvalues for critical systems and forms the problem studied in this work. Our example input geometry consists of an insulated barrel containing a number of hollow rods of fissionable material. Horizontal and vertical cross-sections of

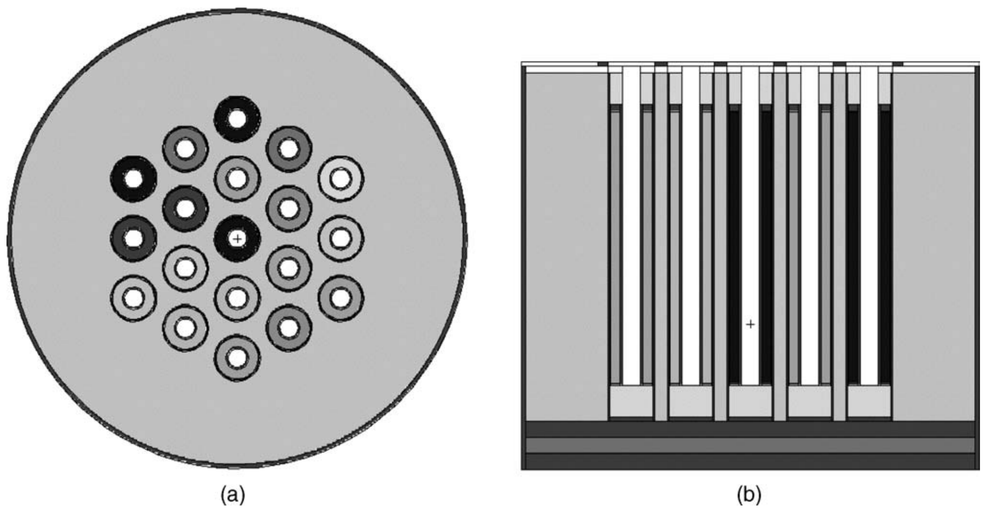

Fig. 1. Horizontal (a) and vertical (b) cross-sections of example geometry.

the geometry are shown in Fig. 1. The shading is used to indicate the different material properties of each rod and also of the insulated barrel. The hollow portion of the rods are indicated as white. The goal of the simulation is to determine if the arrangement of rods is safe, i.e. non-critical. Previous studies on the scalability of parallel Monte Carlo eigenvalue calculations have been rather general in nature [16]. The performance model developed here is both detailed and parametric with both application characteristics (e.g. input geometry, problem size), and system characteristics (e.g. communication latency, bandwidth, achieved processing rate) serving as input.

The remainder of this paper is organized as follows. In Section 2 we further describe MCNP and the Monte Carlo method of non-deterministic particle transport. In Section 3 we develop our performance model through an examination of MCNP's key data structures and computational flows. Our performance model consists of separate computation and system models as described in Sections 3.1 and 3.2, respectively. Following the validation of the model in Section 3.3 we then use the model in Section 4 to predict the performance of MCNP in various performance scenarios. Finally, we conclude the paper in Section 5. This work represents an extension of work presented in [15].

2. Overview of MCNP

MCNP can trace its roots back to the invention of the Monte Carlo method at Los Alamos during World War II. The Monte Carlo method is generally attributed to Metropolis and Ulam [18] and was one of the first application programs to run on early computers in the 1950s. MCNP is the successor to their work and represents over 450 person-years of development. Version 4C of MCNP was used in this analysis. Interestingly, the "Metropolis algorithm", a dynamic method of generating variables with an arbitrary probability distribution used in many Monte Carlo simulations, was invented at a Los Alamos dinner party in 1953 [17].

Monte Carlo methods in general and MCNP specifically do not solve an explicit equation, but rather obtain answers by simulating the interactions between individual particles and a predefined geometry. The accuracy of the calculation increases in proportion to the number of particles used in the simulation. In general, the error in the calculation reduces as the square root of the number of particles. This is in contrast to deterministic transport methods, the most common of which is the discrete ordinates method, that solve the transport equation directly for the average particle behavior over structured [11,12] or unstructured [19,22] meshes. The input geometry for MCNP consists of a collection of cells defined as combinations of primitive shapes such as planes, cylinders and spheres. The material properties are retrieved from an external library. The behavior of each simulated particle and its interaction with the materials travelled through, as defined by the geometry, are recorded to produce a particle *history*. During this process, statistical information about certain events is gathered in histograms or *tallies*. The interaction of each particle and geometry can result in several events such as neutron/photon scatter, capture, and leakage.

The current parallelization strategy of MCNP requires the geometry to be copied to all processors and thus the complexity of the geometry is constrained by the memory available in a single processing node [4]. Parallelism can be utilized to either solve the same problem faster by sub-dividing the number of simulated particles across all processors (strong scaling), or to give a more accurate simulation by simulating more particles in proportion to the number of processors (weak-scaling). During a cycle of MCNP, each processor simulates a designated set of particles. At the end of a cycle, a single processor merges the results from all other processors during a *rendezvous*. This communication pattern requires several steps and a fairly high degree of coordination. Note that the achievable performance of MCNP is both input sensitive (the cost to simulate a particle depends on the complexity of the geometry and materials used) and output sensitive (the complexity of the output depends on the requested tallies).

3. Analytical performance model of MCNP

MCNP is representative of a general parallel application paradigm known as master–slave (e.g. [23]). In this paradigm, a master process is responsible for dividing the work to be done across a number of slave processes. Work assignments and state information are distributed from master to slaves during a "scatter" phase at the start of each cycle. Once the slaves receive their assignments they may begin their local computation. When the slaves have completed their work, they report their results to the master during a "gather" phase. The master then aggregates the results from all the slaves and a new cycle begins. The "scatter" and "gather" phases may actually consist of a sequence of messages. For MCNP the scatter phase consists of two communications, and the gather phase consists of five communications.

The overall communication pattern of MCNP can be analyzed using task graphs (e.g. [5,21]). In this work the *static* parallel structure of a computation is captured using a static task graph, or STG [1] (Fig. 2(a)). The vertices of the STG can represent computation, communication, control flow, or function calls. In Fig. 2, the shaded nodes designate slave activities and the unshaded nodes represent master activities. Likewise, the edges represent control flow dependences or communication dependences. In Fig. 2 dashed edges indicate communications between processors and solid edges represent control flow and/or data dependence on a processor. Each dashed edge is labeled according to the "stages" of a cycle as listed in Table 1.

The STG can be expanded to a *dynamic task graph* or DTG for a particular instance of the application by evaluating all control flow nodes according to the application inputs and processor count specific to that instance (Fig. 2(b)). The dynamic factors are only known when a particular problem and processor configuration is known. These include the processor count and the number of particles to be simulated per cycle. In the case of MCNP, the important input parameters are the number of processors and the number of particle histories. In Fig. 2, the STG for MCNP is evaluated for an instance utilizing four processors. Notice that the DTG is *acyclic* (i.e., all loops have been removed). For clarity, the DTG also shows the parameter values for communication and computation events.

The DTG in particular makes apparent several key characteristics that affect the performance of MCNP. First, MCNP is well-suited to the master–slave paradigm due to the independent nature of particle simulation. As can be seen in Fig. 2(b), during the work phase of MCNP there is no communication between slaves (i.e., any particle is simulated independently of any other particle). Second, the DTG reveals a weakness of MCNP and the master–slave approach in general: *all communication must go through the master*. Although the work can be done in parallel, the gather phase of the application is effectively serialized. Another problem with the master–slave approach also encoded in the DTG, although less apparent, is that the entire input geometry must be replicated across all processors. This can be inferred from the fact that no communication exists during the work phase of the application.

From this initial analysis, we can predict that the performance of MCNP will scale well up to a medium number of processors until communication costs dominate the execution time and then fall off quickly. Indeed, this situation is inherent to the master–slave approach in general and to parallel Monte Carlo eigenvalue calculations specifically [16]. The first limitation (communication serialization) could be potentially mitigated by altering the manner in which much of the data is reported to the master. The second (data replication) could be overcome by distributing the geometry and allowing communication between slaves, to relay geometry or particle information. However, this would involve major changes in the code.

It is worth noting that the task graph technique is general and can be used to model an arbitrary application. The structure of this particular application happens to be master–slave. The approach presented here will work equally well with other program structures (e.g. red-black stencil, sweep, etc.). As a further example, consider the proposed solution to data replication. This would require some frequency of communication between slaves and/or a central geometry or particle server (possibly distributed) during the work phase. Using the task graph technique these communication events can be included in the analysis in a direct manner, thus altering the final form of the model. These modifications could also be incorporated using an explicit data model. In this case, the data will appear to be local to each processor and the model evaluation will be responsible for determining the cost of all data accesses.

3.1. Computation model

The computation component of the performance model includes only those portions of the parallel activity that significantly contribute to the overall execution time. The main stages of a cycle of MCNP corresponding to those depicted in Fig. 2 are listed in Table 1.

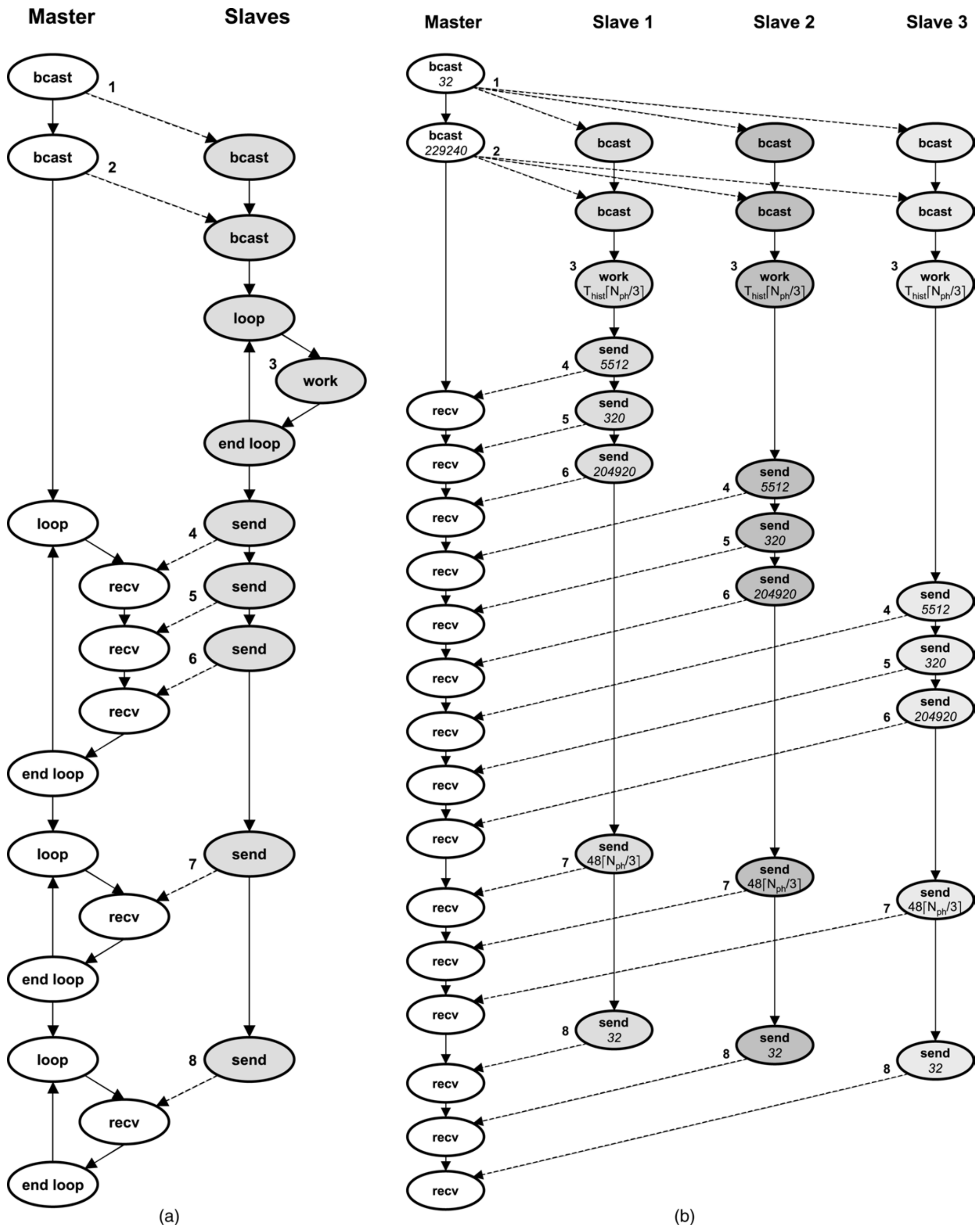

Fig. 2. Static (a) and dynamic (b) task graphs for MCNP.

Table 1
Summary of the parallel activity in one cycle of MCNP

Phase		Source	Action	Weight	Description
Scatter	1	M	bcast	$P*8$	Broadcast particle ranges
	2	M	bcast	229240	Broadcast updates to histories
Work	3	S	work	$T_{\text{hist}} * \left\lceil \frac{N_{\text{ph}}}{P-1} \right\rceil$	Slaves compute particle histories
Gather	4	S	pt2pt	5512	Slaves report common task data
	5	S	pt2pt	320	Slaves report individual tally data
	6	S	pt2pt	204920	Slaves report first task array
	7	S	pt2pt	$48 * \left\lceil \frac{N_{\text{ph}}}{P-1} \right\rceil$	Slaves report second task array
	8	S	pt2pt	32	slaves report timing data

M: master, S: slave(s).

The table summarizes the event source (either master or slave), its type (either collective broadcast, point-to-point communication, or computation), and also the weight associated with the event for each stage of the cycle. The weight is in bytes for all communication events (message sizes), and in terms of the number of particle histories for the computation events. The values determine the weight of the nodes and edges in Fig. 2. The sizes of some of the messages are dependent upon the actual problem being solved. These sizes must be measured prior to the use of the model.

The first task array message (stage 6 in Table 1) is constant for each input geometry and the requested tallies. The tally data message (stage 5) can be calculated simply as the word size (8) times 2 plus the number of requested tallies (38 for this problem). The constant 48 involved in the task array 2 message (stage 7) was obtained by run-time measurement. This is related to the average number of collisions experienced by each particle. A good approximation of this constant can be obtained from a small test run of the code.

The execution time for a single cycle of MCNP can be modeled as:

$$\begin{aligned} T_{\text{total}}(P, N_{\text{ph}}, T_{\text{hist}}) &= T_{\text{scatter}}(P) + T_{\text{slave}}(P, N_{\text{ph}}, T_{\text{hist}}) \\ &\quad + T_{\text{gather}}(P, N_{\text{ph}}), \end{aligned} \tag{1}$$

where the cycle time, T_{total}, is a summation of the scatter, work, and gather phases—T_{scatter}, T_{slave}, and T_{gather}, respectively. The form of this model is additive since the gather and scatter stages are in general synchronized by the bottleneck caused by the master, and the serialization of the communication from the slaves to the master. Each cycle begins with a scatter phase:

$$T_{\text{scatter}}(P) = T_{\text{bcast}}(P*8, P) + T_{\text{bcast}}(229420, P), \tag{2}$$

where the time to perform the collective broadcast operation, $T_{\text{bcast}}(S, P)$, is the time taken to broadcast S bytes across P processors on the target system. The scatter phase corresponds to the first two stages in Table 1.

The work phase, performed on each slave, can be modeled as:

$$T_{\text{slave}}(P, N_{\text{ph}}, T_{\text{hist}}) = \left\lceil \frac{N_{\text{ph}}}{P-1} \right\rceil * T_{\text{hist}}, \tag{3}$$

where N_{ph} is the number of particle histories per cycle which are divided amongst the $P-1$ slave processors. In general, it is more accurate to take the computation time for the slowest slave. However, since each slave is responsible for an equal number of particles, we assume that all slaves will take the same time. The time to perform a single particle history, T_{hist}, can be measured on a single processor for the problem being solved.

The gather phase can be modeled as:

$$\begin{aligned} T_{\text{gather}}(P, N_{\text{ph}}) &= \sum_{i=1}^{P-1} \Bigg(T_{\text{pt2pt}}(5512, i, 0) + T_{\text{pt2pt}}(320, i, 0) \\ &\quad + T_{\text{pt2pt}}(204920, i, 0) \\ &\quad + T_{\text{pt2pt}}\left(48 * \left\lceil \frac{N_{\text{ph}}}{P-1} \right\rceil, i, 0\right) \\ &\quad + T_{\text{pt2pt}}(32, i, 0) \Bigg), \end{aligned} \tag{4}$$

where $T_{\mathrm{pt2pt}}(S, \mathrm{src}, \mathrm{dest})$ is the time required to communicate a message of size S bytes from processor src to processor dest. The five point-to-point communications, listed as stages 4–8 in Table 1, are effectively performed in a serialized way for all slaves due to the master bottleneck. However, an examination of the current messaging within MCNP indicates that some of the data transferred between slaves and the master (specifically stages 4, 5, part of 6, and 8 in Table 1) can be implemented as collective reductions. If we assume that all of stages 4, 5, and 8 as well as half of stage 6 can be implemented as reductions, Eq. (4) can be re-written as

$$\begin{aligned} T_{\mathrm{gather}}(P, N_{\mathrm{ph}}) &= \sum_{i=1}^{P-1} \left(T_{\mathrm{pt2pt}}(102460, i, 0) \right. \\ &\quad \left. + T_{\mathrm{pt2pt}}\left(48 * \left\lceil \frac{N_{\mathrm{ph}}}{P-1} \right\rceil, i, 0\right)\right) \\ &\quad + T_{\mathrm{reduce}}(5512, P) + T_{\mathrm{reduce}}(320, P) \\ &\quad + T_{\mathrm{reduce}}(102460, P) + T_{\mathrm{reduce}}(32, P), \qquad (5) \end{aligned}$$

where $T_{\mathrm{reduce}}(S, P)$ is the time required to perform a reduction of S bytes using P processors.

3.2. System model

The computation model as formulated in Section 3.1 requires components in the system model for point-to-point communication times, collective broadcast times, collective reduction operations, the time required to perform a single particle history, and also the memory performance of a single node (for communication packing). MCNP actually uses the UPS messaging library [2] for communication between processors. UPS provides a generic interface with a retargetable back-end. It allows a message of arbitrary length to be built from many smaller variables using packing functions in a similar way to that of PVM—a feature that is heavily utilized in MCNP. In the analysis that follows a 32-node AlphaServer ES40 cluster is used as the experimental testbed. This machine has four processors per node with a total machine size of 128 processors. Each processor is an EV68 running at 833 MHz, having an 8 MB level 2 cache. Each node has 8 GB of main memory. The nodes are connected using the Quadrics QsNet-1 high-performance network [20]. This network boasts high-performance communication with a typical MPI latency of 5 μs and a throughput of up to 340 MB/s per communication direction.

Table 2
Summary of system model parameters for the AlphaServer ES40 (S in bytes)

$T_{\mathrm{pack}}(S)$ (ns)	$L_c(S)$ (μs), $B_c(S)$ (MB/s)	T_{hist} (s)
0.12, $S < 32K$	5.05, 0.0, $S < 64$	7.98×10^{-4}
0.16, $32K \leq S \leq 4M$	5.47, 78, $64 \leq S < 512$	
0.67, $S > 4M$	10.3, 294, $S \geq 512$	

Measured MPI latency and bandwidth for inter-node unidirectional communication (point-to-point) were obtained using in-house micro-benchmarks. The collective broadcast and reduction operations for P processors can be assumed to take at most $\log_2(P)$ times that of a single point-to-point communication. The communication costs also include packing operations, implemented in UPS. The point-to-point, broadcast, and reduction communication operations are modeled as

$$T_{\mathrm{pt2pt}}(S, \mathrm{src}, \mathrm{dest}) = T_{\mathrm{pack}}(S) + L_c(S) + S/B_c(S), \qquad (6)$$

$$T_{\mathrm{bcast}}(S, P) = T_{\mathrm{pt2pt}}(S) * \log_2(P), \qquad (7)$$

$$T_{\mathrm{reduce}}(S, P) = T_{\mathrm{pt2pt}}(S) * \log_2(P), \qquad (8)$$

where S is the size of the message in bytes, $T_{\mathrm{pack}}(S)$ the time to pack S bytes, $L_c(S)$ and $B_c(S)$ the latency and bandwidth of a message of size S bytes. The parameters used in the system model are summarized in Table 2 for the AlphaServer ES40 cluster.

3.3. Performance model validation

The model is validated against measurements made on our testbed system showing high accuracy—approximately 10% error across all cases. Measurements and predictions are shown in Fig. 3 for seven sets of particle histories per cycle ($N_{\mathrm{ph}} = 100$, 500, 1000, 5000, 10,000, 50,000, and 100,000) on a range of processor counts (a strong-scaling analysis). The geometry is the same for all runs. Note that for small N_{ph}, the communication costs soon dominate the processing time, resulting in poor scalability. For large N_{ph}, the

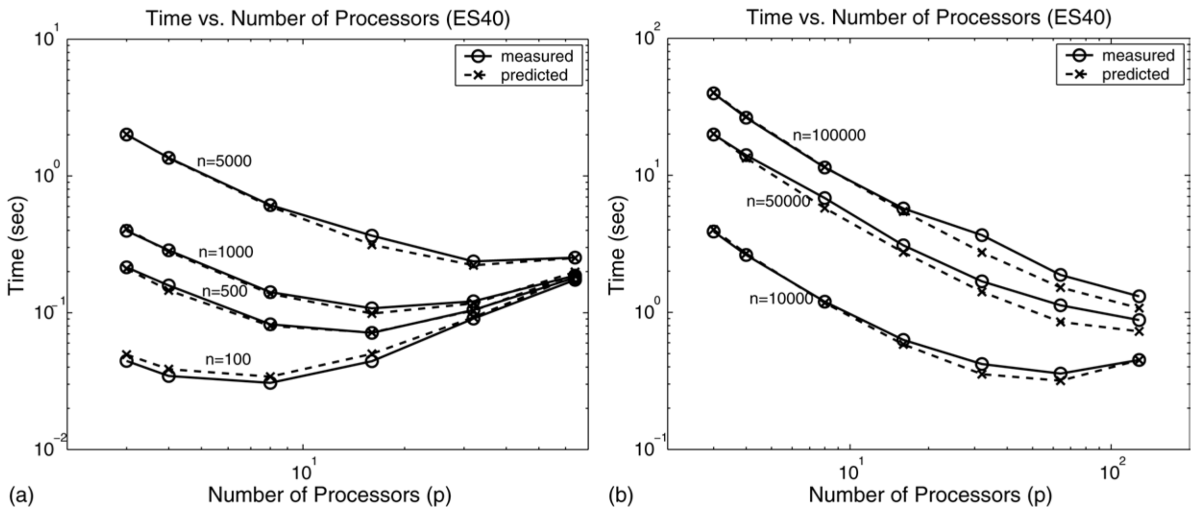

Fig. 3. Measured and predicted times for small (a) and large (b) inputs.

scalability is better up to a higher processor count. Also note that the performance characteristics correspond to our task graph analysis in Section 3. The performance of MCNP indeed scales very well up to a moderate number of processors and then falls off quickly once communication costs dominate the execution time.

4. Performance study of MCNP

Once the model has been validated and shown to provide reasonable prediction accuracy, the model may be used to explore the performance of the application on systems and configurations that are not possible to be measured. In this section the model is used to explore the performance of MCNP on larger configurations of current ES40 systems. It is also used to explore the possible improvements resulting from refining the code to make use of reduction operations as suggested in Section 3.1. The model is further utilized to investigate the performance of future systems assuming performance improvements in individual sub-system characteristics such as latency, bandwidth, and processor speed.

4.1. Scaling behavior of larger systems

The MCNP performance model is used to explore the expected performance on larger AlphaServer ES40 systems in Fig. 4 for both strong and weak-scaling modes. As the processor count increases in the strong scaling mode, the amount of work per slave decreases and hence communication costs soon become a significant percentage of the overall run-time. In weak-scaling, as the processor count increases the amount of computation per processor is constant and thus the overall run-time increases more gradually due to increased communication costs. Overall it can be seen that in a strong-scaling mode, MCNP scales up to 512 processors on the problem being studied for $N_{\mathrm{ph}} = 1 \times 10^6$. In weak-scaling, the performance of MCNP is much better and actually scales up to 8192 processors for $N_{\mathrm{ph}} = P \times 10^3$.

The component time information is also shown in Fig. 5 for both the strong-scaling and weak-scaling modes. The major contribution to the total time changes from primarily computation for low numbers of processors to primarily communication for large numbers of processors. It is also clear that one message (task array 1) dominates the communication time.

4.2. Performance prediction of future systems

Using the model, the expected performance of MCNP can be examined in advance of possible code refinements. Here we examine the case of the code modifications of using reduction operations in the gather phase. Fig. 6 shows the expected performance im-

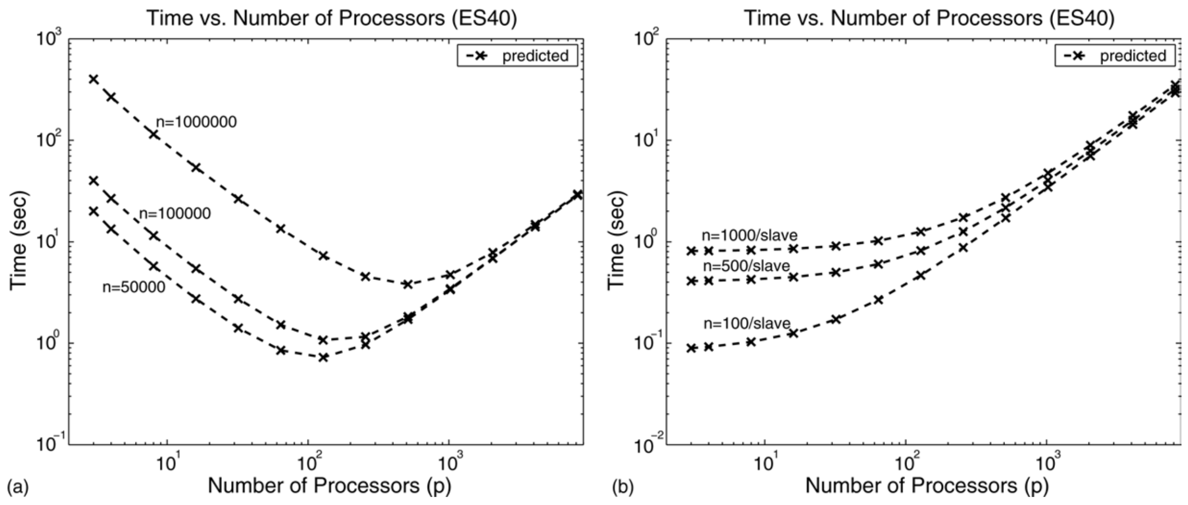

Fig. 4. Expected cycle times on ES40 systems for strong (a) and weak (b) scaling.

provement for MCNP in both strong-scaling and weak-scaling modes. It can be seen that the scalability is improved by approximately a factor of two for the highest processor count. This indicates that should the code be modified then there will be a beneficial impact on its scalability.

The impact of system performance improvements on the run-time of MCNP can also be quantified in advance of such systems being available. Here we examine a number of what-if scenarios by considering the performance of the communication and computation sub-systems to be improved individually by a factor of 8. This is achieved by modifying the parameters in the system model. The factor of 8 was chosen to be indicative of what may happen to these sub-system performances over the next 5 years. The relative improvement over the existing ES40 system is shown in Fig. 6. Also included in Fig. 6 is the relative perfor-

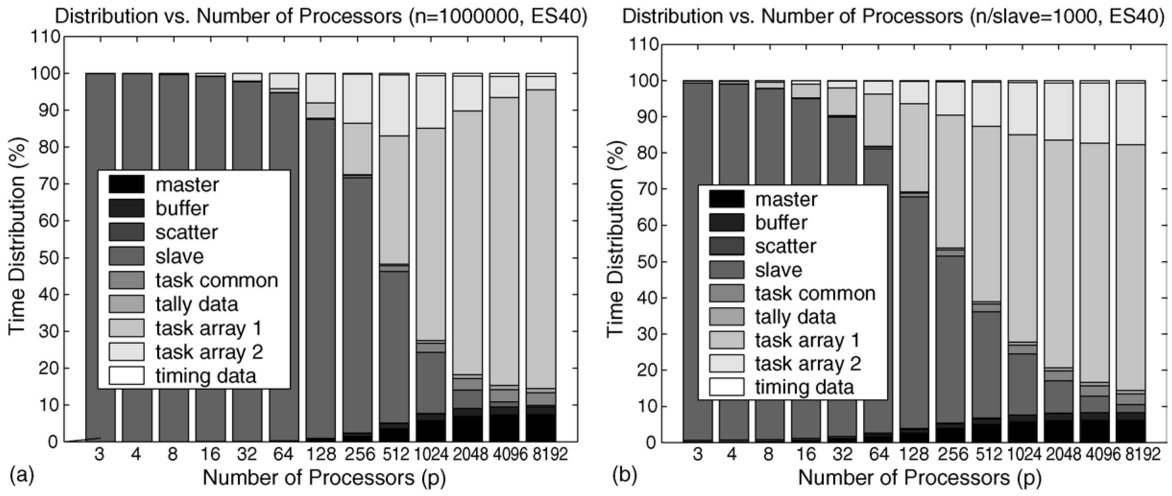

Fig. 5. Component time predictions for MCNP on larger AlphaServer ES40 systems under strong (a) and weak (b) scaling.

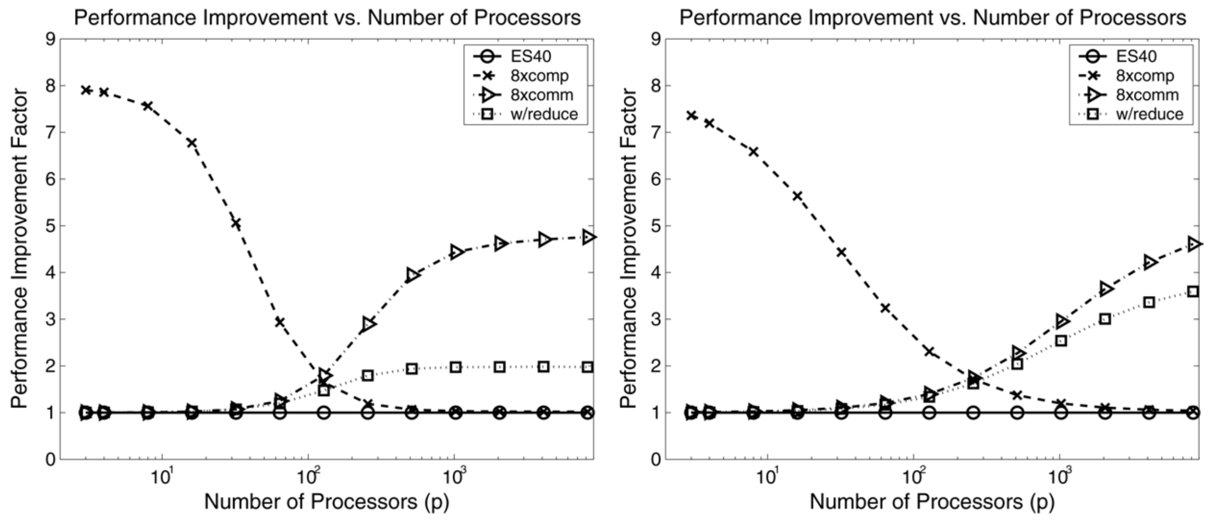

Fig. 6. Relative impact of improved computation and communication capabilities for strong (a) and weak (b) scaling.

mance improvement that could be obtained if just the modifications to the code as described in Section 3.1 were implemented.

It can be seen from Fig. 6 that an increase in computation capability has a much greater impact on performance for small numbers of processors, but rapidly declines as the number of processors is increased. Similarly on larger processor counts, the increase in communication capability will have a larger impact (due to communication constituting a larger percentage of the execution time as the number of processors increases). Various other performance scenarios can be examined in a similar way.

5. Conclusion

In this work we have developed a detailed analytical performance model for MCNP. The model includes the main code characteristics and separates out the computation and system characteristics. The model is based on a static analysis of the application but is parameterized in terms of its dynamic behavior. Through a validation process on a 32-node AlphaServer ES40 cluster, we have shown the model to be accurate with a typical error of 10%.

The model has been used to explore a number of performance scenarios. In a scalability analysis, the model was used to give expected performance on larger ES40 systems. This analysis showed that in a weak-scaling mode the application will scale to thousands of processors whereas in a strong-scaling mode the application scales to only hundreds of processors.

The performance of MCNP was also examined for the case of modifying the communication structure in the application to include the use of collective reductions. This analysis indicated that the performance could be improved on large processor counts if such modifications were implemented. In addition, the performance of MCNP was examined on a number of hypothetical systems which included faster computation or communication sub-systems. It was shown that increases in computation speed have the greatest effect on smaller processor counts, and increases in communication speed have the greatest effect on larger processor counts.

Through these analyses the benefits of developing a performance model of an application have been illustrated. Once a model has been validated it can be used to predict performance on systems or configurations that cannot be measured. The model has been used to analyze many scenarios in this work, and will be used to explore the performance on future machines as they become available. The model is part of an ongoing effort to model the ASC workload and complements existing models for deterministic particle transport on structured and unstructured meshes and for adaptive mesh refinement applications.

Acknowledgements

This work was supported in part by LDRD 2001609DR "Performance Analysis and Modeling of Extreme-Scale Parallel Architectures". Mark Mathis is currently a PhD student at Texas A&M University under the guidance of Nancy Amato and is supported in part by a Department of Energy High Performance Computer Science Fellowship. Los Alamos National Laboratory is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36.

References

[1] V.S. Adve, M.K. Vernon, Parallel program performance prediction using deterministic task graph analysis, ACM Trans. Comput. Syst. 22 (1) (2002) 94–136.
[2] R. Barrett, M. McKay, UPS: Unified Parallel Software User's Guide and Reference Manual, Los Alamos National Laboratory, 2002.
[3] J.F. Briesmeister, MCNP™—A General Purpose Monte Carlo N-Particle Transport Code, Version 4C, Technical Report LA-13709-M, Los Alamos National Laboratory, April 2000.
[4] L.J. Cox, DMMP Upgrade for MCNP4C™, Los Alamos National Laboratory Research Note, April 2001.
[5] E.F. Gehringer, D.P. Siewiorek, Z. Segall, Parallel Processing: The Cm* Experience, Digital Press/DEC, Bedford, MA, 1986.
[6] A. Hoisie, O. Lubeck, H.J. Wasserman, Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications, Int. J. High Perform. Comput. Appl. (HPCA) 14 (4) (2000) 330–346.
[7] D.J. Kerbyson, H.J. Alme, A. Hoisie, F. Petrini, H.J. Wasserman, M.L. Gittings, Predictive performance and scalability modeling of a large-scale application, in: Proceedings of the IEEE/ACM Supercomputing, Denver, November 2001.
[8] D.J. Kerbyson, A. Hoisie, S.D. Pautz, Performance modeling of deterministic transport computations, in: Performance Analysis and Grid Computing, Kluwer, 2003, pp. 21–39.
[9] D.J. Kerbyson, H.J. Wasserman, A. Hoisie, Exploring advanced architectures using performance prediction, in: Innovative Architecture for Future Generation High-performance Processors and Systems, IEEE CS Press, 2002, pp. 27–37.
[10] D.J. Kerbyson, A. Hoisie, H.J. Wasserman, Veryfying large-scale system performance during installation using Modeling, in: High Performance Scientific and Engineering Computing Hardware/Software Support, Kluwer, 2003, pp. 43–156.
[11] K.R. Koch, R.S. Baker, R.E. Alcouffe, Solution of the first-order form of the 3D discrete ordinates equation on a massively parallel processor, Trans. Am. Nucl. Soc. 65 (108) (1992) 198–199.
[12] E.E. Lewis, W.F. Miller, Computational Methods of Neutron Transport, American Nuclear Society Press, LaGrange Park, IL, 1993.
[13] M.M. Mathis, N.M. Amato, M.L. Adams, A general performance model for parallel sweeps on orthogonal grids for particle transport calculations, in: Proceedings of the ACM International Conference on Supercomputing (ICS), Santa Fe, May 2000, pp. 255–263.
[14] M.M. Mathis, D.J. Kerbyson, Performance modeling of unstructured mesh particle transport calculations, in: Proceedings of the International Parallel and Distributed Proceedings Symposium (IPDPS), Santa Fe, April 2004.
[15] M.M. Mathis, D.J. Kerbyson, A. Hoisie, A performance model of non-deterministic particle transport on large-scale systems, in: Proceedings of the International Conference on Computational Science (ICCS), Part 3, Melbourne, June 2003.
[16] S. Matsuura, R.N. Blomquist, F.B. Brown, Parallel Monte Carlo eigenvalue calculations, Trans. Am. Nucl. Soc. 71 (1994) 199–202.
[17] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculation by fast computing machines, J. Chem. Phys. 21 (1953) 1087–1092.
[18] N. Metropolis, S. Ulam, The Monte Carlo method, J. Am. Stat. Assoc. 44 (1949) 335–341.
[19] S.D. Pautz, An algorithm for parallel S_N sweeps on unstructured meshes, J. Nucl. Sci. Eng. 140 (2) (2002) 111–136.
[20] F. Petrini, W.C. Feng, A. Hoisie, S. Coll, E. Frachtenberg, The quadrics network: high-performance clustering technology, IEEE Micro 22 (1) (2002) 46–57.
[21] K. Pingali, M. Beck, R. Johnson, M. Moudgill, P.L. Stodghill, Dependence flow graphs: an algebraic approach to program dependencies, in: Proceedings of the ACM Symposium on Principles of Programming Languages (POPL), Orlando, 1991, pp. 67–78.
[22] S. Plimpton, B. Hendrickson, S. Burns, W. McLendon, Parallel algorithms for radiation transport on unstructured grids, in: Proceedings of the IEEE/ACM Supercomputing, Dallas, November 2000.
[23] J.M. Schopf, Structural prediction models for high-performance distributed applications, in: Proceedings of the Cluster Computing Conference, Atlanta, March 1997.

Mark M. Mathis is a PhD candidate in the Parasol Laboratory at Texas A&M University. His interests lie in high performance computing systems, performance modeling and analysis. Mr. Mathis is an inaugural recipient of the Department of Energy High Performance Computer Science Fellowship and recent collaborator with researchers at Los Alamos National Laboratory.

Darren Kerbyson is currently a researcher in the Performance and Architecture Lab at Los Alamos. Prior to this he was a senior Lecturer in Computer Systems at the University of Warwick in the UK. He has been active in the areas of performance modeling, parallel and distributed processing systems, and image analysis for the last 15 years. He has worked on many performance orientated projects funded by the European Esprit program, UK Government, ONR, and DARPA. He has published over 70 papers in these areas and has taught courses at undergraduate and postgraduate levels as well as supervising numerous PhD students. He is currently involved in the modeling of large-scale applications on current and future supercomputers at Los Alamos. He is a member of the ACM and the IEEE.

Adolfy Hoisie is a Staff Scientist and the group leader of the Modeling, Algorithms, and Informatics Group in the Computer and Computational Sciences Division at LANL. He also leads the Performance and Architectures Laboratory. From 1987 until he joined LANL in 1997, he was a researcher at Cornell University. His area of research is performance evaluation of high-performance architectures. He has published extensively, lectured at numerous conferences and workshops, often as an invited speaker, and taught tutorials in this field at important events worldwide. He was the winner of the Gordon Bell Award in 1996, and co-author to the recently published SIAM monograph on performance optimization.